

BFS

Breadth-First Search (BFS) is a graph traversal algorithm that explores nodes level by level. It starts from a chosen source node, visits all the nodes directly connected to it first, and then moves on to the neighbors of those nodes, continuing outward like waves. To keep track of the order in which nodes should be visited, BFS uses a queue (FIFO). When a node is visited, it is marked as “visited” and its unvisited neighbors are added to the queue. BFS works for both graphs and trees and is especially useful for finding the shortest path in an unweighted graph, checking if a graph is bipartite, and performing level-order traversal in trees. Its time complexity is $O(V + E)$, where V is the number of vertices and E is the number of edges, and it generally requires $O(V)$ space for storing the queue and visited set.

Advantages

- **Finds shortest path** in an unweighted graph because it explores by levels.
- **Guaranteed completeness:** if a solution exists, BFS will find it.
- **Simple and easy to implement** using a queue.
- **Works well for level-order traversal** in trees.
- **Useful for checking bipartite graphs**, connected components, and network spreading.

Limitations

- **High memory usage:** BFS stores many nodes in the queue, especially for wide graphs.
- **Not suitable for very large or infinite search spaces**, as memory can grow exponentially.
- **Slower than DFS** when the solution is far from the root.
- **Does not work for weighted graphs** (unless all weights are equal).

Steps of BFS

Step 1 — Initialize

- Create a **queue** (FIFO).
- Create a **visited set/array**.
- Mark **1** as **visited**.
- Enqueue **1**.

Queue: [1]

Visited: {1}

Step 2 — Dequeue and Explore Node 1

- Dequeue front element → **1**
- Visit node **1**.
- Check neighbors of **1** → 2, 3

For each neighbor:

- **2** not visited → mark visited and enqueue
- **3** not visited → mark visited and enqueue

Queue: [2, 3]

Visited: {1, 2, 3}

Order so far: 1

Step 3 — Dequeue and Explore Node 2

- Dequeue → **2**
- Visit node **2**
- Neighbors of 2 → 1, 4, 5

Check each neighbor:

- **1** is already visited → skip
- **4** not visited → visit + enqueue
- **5** not visited → visit + enqueue

Queue: [3, 4, 5]

Visited: {1, 2, 3, 4, 5}

Order so far: 1, 2

Step 4 — Dequeue and Explore Node 3

- Dequeue → **3**
- Visit node **3**

- Neighbors of 3 → 1, 6

Check each:

- 1 visited → skip
- 6 not visited → visit + enqueue

Queue: [4, 5, 6]

Visited: {1, 2, 3, 4, 5, 6}

Order so far: 1, 2, 3

Step 5 — Dequeue and Explore Node 4

- Dequeue → 4
- Visit node 4
- Neighbors of 4 → 2
- 2 is already visited → skip

Queue: [5, 6]

Order so far: 1, 2, 3, 4

Step 6 — Dequeue and Explore Node 5

- Dequeue → 5
- Visit node 5
- Neighbors → 2
- 2 visited → skip

Queue: [6]

Order so far: 1, 2, 3, 4, 5

Step 7 — Dequeue and Explore Node 6

- Dequeue → 6
- Visit node 6
- Neighbors → 3
- 3 visited → skip

Queue: []

Order so far: 1, 2, 3, 4, 5, 6

Step 8 — Queue Empty → BFS Complete

All reachable nodes are visited.

Pseudocode

BFS(graph, start):

 create a queue Q

 mark start as visited

 enqueue start into Q

 while Q is not empty:

 node = dequeue Q

 for each neighbor of node:

 if neighbor is not visited:

 mark neighbor as visited

 enqueue neighbor

Time Complexity

- **Time:** $O(V + E)$
 - V = number of vertices
 - E = number of edges
- **Space:** $O(V)$
 - for queue + visited array

BFS code:

<https://github.com/pulokdas062/Graph-Algorithm/blob/main/BFS/BFS%20basic/Basic.cpp>

